TR-IT-0085

CNTR: Basic Functions for Centering Experiments with ASURA

Mark Seligman

December, 1994

Abstract

Centering is a family of theories of cohesion, the linkage among utterances. Entities (especially those expressed by noun phrases) are tracked through successive utterances, with empahsis on entities which are especially salient because of their syntactic prominence. Such tracking enables a classification of utterance-to-utterance transitions, ranked by respective cohesiveness. By conmputing all possible interpretations of the current utterance and selecting the one which would provide the most cohesive transition, it is possible to resolve the reference of many pronouns, including zero pronouns. This report describes CNTR, an implementation of a representative centering approach using ASURA's transfer system, based on suggestions by Nagata. Full code and programmer-oriented instructions are provided, as well as suggestions for future experiments. The mechanisms described here are fully compatible with pronoun resolution techniques already implemented in ASURA, especially Dohsaka's use of honorific information.

ATR音声翻訳通信研究所
ATR Interpreting Telecommunications Research Laboratories

Abstract

Centering is a family of theories of cohesion, the linkage among utterances. Entities (especially those expressed by noun phrases) are tracked through successive utterances, with empahsis on entities which are especially salient because of their syntactic prominence. Such tracking enables a classification of utterance-to-utterance transitions, ranked by respective cohesiveness. By conmputing all possible interpretations of the current utterance and selecting the one which would provide the most cohesive transition, it is possible to resolve the reference of many pronouns, including zero pronouns. This report describes CNTR, an implementation of a representative centering approach using ASURA's transfer system, based on suggestions by Masaaki Nagata. Full code and programmer-oriented instructions are provided, as well as suggestions for future experiments. The mechanisms described here are fully compatible with pronoun resolution techniques already implemented in ASURA, especially Dohsaka's use of honorific information.

1 Introduction

When we want to find the reference for a pronoun, or zero pronoun, or definite NP, we need to know which entities (generally noun phrase) have been mentioned until now in the discourse and how salient they are. That is, we need models of short- and long-term memory and attention.

Currently, the dominant approach to modeling *short*-term memory and attention is *centering*. It tells us which entities among the recently mentioned entities are most salient, or likely to receive attention. "Recent" here means "in the last few utterances". Because these recent and salient entities should be accessible for both the speaker and the listener, the speaker assumes that the listener can identify them easily. So when the speaker wants to refer to them in the next utterance, s/he will probably not use a full form; instead, s/he will use reduced forms like pronouns or zero pronouns. One of the salient entities will become the theme ("backward-looking center") of the next utterance – that is, its main connection to the preceding discourse, the known entity which the utterance will give new information about.

How do we know which entities in short-term memory are most salient, and thus most likely to be expressed in reduced form in the next utterance, or most likely to become its theme? By using a hierarchy of salience, which for instance can say that a grammatical topic (marked with wa) is more salient than various sorts of object (marked with *wo, ni*, etc.). We also can observe how salience tends to change from utterance to utterance – that is, what sorts of transitions are most likely.

While centering is the most established approach to pronoun resolution, it is certainly not the only one. There are several specialized approaches, such as Dohsaka's use of honorific information to resolve Japanese zeros [Dohsaka 1990].

A few other examples of specialized techniques are provided by [Nasukawa 1994]. This paper applies the two techniques below to Japanese texts:

- Collocation (modifiee-modifier) relationships can be extracted from the text, even without the use of a full parse. Having seen a sentence *He moved his residence*, we learn that residence can be the object of the verb move. We can then use this information as a selectional constraint to resolve the pronoun it in a sentence like *The castle in Camelot remained the residence of the king until 536 when he moved it to London.* Repeated collocation patterns can be recognized and exploited: using a Case Role Persistence Rule, we prefer resolution candidate noun phrases that filled the same role in previous utterances as a reduced form plays in the current one: In *Mary gave an apple to Susan. John also gave her an apple, her* refers to Susan, not Mary. Syntactic parallelism can be used in a similar way: *The girl scout leader paired mary with Susan, but she (the leader) had paired her (Mary) with Nancy last time.*
- The frequency of appearance of a noun phrase in the utterances preceding a reduced form can give an indication of its salience, and thus its priority as a resolution candidate.

Such specialized techniques can be compared with, and combined with, centering-based approaches.

Are centering models effective models of short-term memory and attention and their effects on reference? [Takada and Doi 1994] claim the ability to correctly identify 76 percent - 81 percent of the Japanese pronouns and zero pronouns in certain corpora without the use of a global focusing model [Grosz and Sidner 1986], a 5-6 percent improvement over previous versions. It would be instructive to check these numbers for our corpora. Use of specialized techniques might add several percent above the raw centering scores for Japanese. Below, we compare the approach of [Takada and Doi 1994] with other current techniques.

While centering models are designed to handle reference problems, they also provide part of the information necessary for determining whether noun phrases are *definite* in English, German, and other European languages. While this determination is not as important for communication as pronoun resolution, it provides an additional reason for experimentation with centering-type studies.

To enable experimentation in ASURA, CNTR, a set of prototype programs based on proposals by Masaaki Nagata (personal communication) has been implemented using ASURA's RWS transfer system [Hasegawa 1990]. This report describes the programs and provides theoretical background necessary for understanding them. Full code and programmer-oriented instructions are given, and suggestions are made regarding future experiments.

The mechanisms described here are fully compatible with pronoun resolution techniques already implemented in ASURA, especially Dohsaka's use of honorific information. If work on ASURA is discontinued, the CNTR programs can be used as models when writing similar programs for other parsers.

2 Summary of Centering, Comparison of Approaches

This section reviews basic centering ideas and briefly compares several approaches: [Walker, Iida, and Cote 1992], [Kameyama 1985, 1986, 1988], and recent suggestions by [Takada and Doi 1994].

2.1 Linkage Between Utterances

Centering is a family of theories concerning the linkage, or connection, between utterances. Sequential utterances are linked – and thus potentially coherent – if they both refer to the same discourse entity or entities: for instance, Taroo may be a referent in both U1 and U2. (Entities are usually semantic objects expressed by NPs, but propositions expressing clauses, too, may be handled in future versions of the theory). "Center" is sometimes used as an alternative name for "entity".

2.2 The Meaning of Salience

Entities (or centers) have differing degrees of salience. If entity1 (Taroo) is called more salient than entity2 (Hanako), this means that it can be identified more easily by discourse participants. (The underlying cognitive reason is not made explicit. Perhaps a salient entity is more active in short term memory or more accessible in long-term memory.)

2.3 Degrees Of Salience

Salience may be indicated in several ways by language producers, but the most important indicator for current theories is grammatical function. For example, in [Kameyama 1986] the following (incomplete) hierarchy is important for Japanese: (TOPIC (WA) > SUBJECT (WA/GA) > OBJECT (WO)/OBJECT2 (NI) > OTHERS, i.e. oblique, possessor, etc.) It may also be important to know whether or not an entity has already been expressed as a reduced form (pronoun or zero), since only very salient entities are expressed in this way.

2.4 Cp And Cb

The most salient entity in the current utterance Ui is called its preferred center, or Cp. The most salient entity in Ui-1 (the last utterance) which occurs in Ui is called its backward-looking center, or Cb. Among all of the links or shared entities between the current utterance (Ui) and the last one (Ui-1), the Cb is the entity in the current utterance (Ui) that was the most salient or easiest to identify in the last utterance (Ui-1). We also need a way to refer to the full set of entities in an utterance: they are the Cf, or forward-looking centers. Cf members may become referents of the next utterance. (There is some disagreement about terminology: in [Kameyama 1985, 1986] and [Takada and Doi 1994], Cf refers to the full set minus the Cb.)

2.5 The Relation Between Salience And Linkage

As mentioned, centering theory concerns linkage between consecutive utterances. More exactly, it concerns the relation between salience and linkage. For instance, will the most salient entity in this utterance be the most salient in the next utterance, too? (Will the Cp of Ui – which is by definition the Cb of Ui+1 – also be the Cp of Ui+1?)

This question is important in all recent versions of the theory, though the terminology varies from version to version.

2.6 Transitions And Their Likelihood

We call a sequence of two utterances a transition. Some transition types apparently give a stronger sense of continuity or coherence than others, and in this sense are more expected.

For instance, as we will see below, a transition in which the most salient entity in both utterances is the same entity, (Cb(Ui) = Cp(Ui)), is apparently more canonical, and felt by listeners to be easier to interpret, than a sequence in which this condition fails.

One can observe several such factors and their combinations to give a scale of transition likelihoods.

2.7 Application To Reference Resolution

When several reduced forms are used in an utterance, their reference must be resolved. In the worst case, the number of possible interpretations could be equal to the number of permutations of reduced forms against possible referents from the previous utterance. (Note: Centering theories themselves have focused on the relation between two consecutive utterances. Long-distance reference requires a separate theory, e.g. of global focusing.) Actually, constraints on co-reference and semantic selectional constraints permit the elimination of some candidate combinations, but multiple possible combinations may remain.

However, transition likelihoods can rank any remaining interpretations: e.g., "Under Interpretation1, the most salient entity in both utterances would be the same, (Cb(Ui) = Cp(Ui)); but this is not so under Interpretation2. Thus Interpretation1 is more likely."

We have now presented the main elements of (generic) centering theory and its application to reference resolution. We can go on to briefly compare three approaches.

2.8 Three Centering Approaches Compared

Walker et al In [Walker et al 1992], transitions are considered mainly in terms of two factors:

- (a) Is Cb(Ui) equal to Cp(Ui)? Does the most salient entity remain the same from utterance to utterance? Recall that the Cb of Ui indicates the most salient entity of Ui-1 which appears in Ui; and Cp of Ui is the most salient entity of Ui.

  The first step in shifting Cbs – that is, in shifting a discourse "about" entity1 and making it "about" entity2 – is to make a transition in which the most salient entity of Ui-1 (the previous utterance) and Ui (the current utterance) are not the same.

- (b) Is Cb(Ui-1) equal to Cb(Ui)? Does the backward-looking center remain the same from utterance to utterance? As just described, if the most salient entity shifts between e.g. U1 and U2, then between U2 and U3 the discourse link (Cb) will shift. (This follows from the definition of Cb: it is the most salient entity in Ui-1 which is also in Ui.) The link between U1 and U2 – the most salient entity in U1 which is also in U2, the Cb of U2 – will not be the same as the link between U2 and U3, the Cb of U3. In this way, a discourse which has been "about" entity1 (because entity1 is the link during transition U1-U2) becomes a discourse "about" entity2 (because entity2 is the link during transition U2-U3).

By permuting these factors, we get four possible transition types, presented in order of likelihood:

- **Continue:** (a yes, b yes) Discourse link, Cb, continues during the transition, and that entity remains the most salient in the second utterance. In the next transition, the Cb will remain the same: the discourse will continue to be about the entity which has been most salient in both utterances of the Continue transition.

- **Retain:** (a no, b yes) Discourse link, Cb, continues during the transition, but that entity is no longer the most salient in the second utterance. In the next transition, the Cb will shift: the discourse will then be about the entity which moved into the most salient position of the second utterance of the Retain transition.

- **Smooth-Shift:** (a yes, b no) (Following a Retain transition) The discourse link, Cb, shifts between the first and second utterance. Furthermore, the most salient entity in U1 continues to be the most salient in U2.

- **Shift:** (a no, b no) (Following a Retain transition) The discourse link, Cb, shifts between the first and second utterance. However, the most salient entity in U1 is not the most salient in U2.

Here's an example showing all four transition types. This saliency hierarchy is used: TOPIC > EMPATHY > SUBJ > OBJ2 > OBJECT > OTHERS

(EMPATHY is an indication of viewpoint in Japanese utterances with kureru, ageru, etc. For instance, in *Taroo ga ziroo ni hon wo kureta*, Ziroo is the locus of empathy. Kameyama notes that this terminology is misleading since it suggests an emotional involvement, and substitutes the term IDENT, for identification.)

In the examples, possible interpretations appear as Cf listings, with entities in salience order. Notice that two interpretations are possible for the final utterance. Since Smooth-Shift transitions are preferred over Shift transitions for Walker et al, they predict that the first interpretation will be preferred – and 32 out of 34 informants agreed.

```
(a) Taroo ga kooen de hon wo yondeimashita.
Taroo SUBJ park at book OBJ reading-was.
Taroo was reading a book in the park.
Cb: [?] Cf: [Taroo/SUBJ, book/OBJ] NO TRANSITION


(b) 0 cola wo kai ni baiten ni hairimashita.
SUBJ cola OBJ buy to shop into entered
He entered a shop to buy a cola.
Cb: [Taroo] Cf: [Taroo/SUBJ, cola/OBJ] CONTINUE

(c) Ziroo wa 0  sokode guuzen dekuwashimashita.
Ziroo TOP/SUBJ OBJ there by chance met.
Ziroo met him there by chance.
Cb: [Taroo] Cf: [Ziroo/TOP, Taroo/OBJ] RETAIN

(d) 0 0  eiga ni sasoimashita.
SUBJ OBJ movie to invited
He invited him to a movie.
Cb: [Ziroo] Cf1: [Ziroo/SUBJ, Taroo/OBJ] SMOOTH-SHIFT (32 people)
            Cf2: [Taroo/SUBJ, Ziroo/OBJ] SHIFT (2 people)
```

After presenting their treatment of transitions, Walker et al discuss the relative salience of TOPIC and EMPATHY in Japanese. (To handle certain ambiguities, they also propose a Zero Topic Assignment rule which is beyond the scope of this short review.)

Kameyama An earlier treatment of transitions was that of [Kameyama 1986, 1988]. Kameyama's suggestion is that the interpretation of Japanese zeros depends on a default preference hierarchy of the properties to be shared between the antecedent and the zero. This property-sharing constraint is as follows:

Two zero-pronouns in adjacent utterances which both specify the same discourse entity must share one of the following properties (in descending order of preference): 1) both IDENT and SUBJECT, 2) IDENT alone, 3) both NONIDENT and NONSUBJECT, 5) NONSUBJECT alone, or 6) NONIDENT alone. (IDENT is Kameyama's preferred term for EMPATHY – an indication of viewpoint in sentences with kureru, ageru, etc.)

Sentences with one and only one zero receive special treatment: there is a default preference order among full NPs considered as the potential antecedent: TOPIC > IDENT > SUBJECT > OBJECT(S) > OTHERS.

Walker et al describe themselves as building on top of this earlier work. Kameyama's formulation, they believe, covers many of their examples if one assumes that SUBJECTS become EMPATHY loci by default. However, they discuss several types of sequences in which Kameyama's formulation gives no prediction, or the wrong one.

Takada and Doi In the salience hierarchy of her dissertation work [Kameyama 1985], Kameyama tries to model the effect of prior reduced expression on salience: the Cb of an utterance comes to have special salience, and ranks higher than any grammatical function. However, if there are multiple zero pronouns or referring pronouns (e.g. kanojo) in an utterance, only one of them can become the backward-looking center, thus gaining special salience and priority. In contrast, any other zero or explicit pronouns get no special treatment, and are treated in the same way as fully expressed NPs. [Takada and Doi 1994], however, propose that such reduced forms should actually

be considered more salient than fully expressed forms. Their main point: If an entity is referred to by a reduced form once, this indicates its high salience, so the same entity is increasingly likely to be expressed by a reduced form again.

For example, given a zero pronoun encoding Saburo and an overt form Ziroo encoding the entity Ziroo in some utterance, and given a single zero pronoun in the next utterance which could refer to either Saburo or Ziroo according to the salience hierarchy, the more likely referent is Saburo.

To implement this suggestion, rather than a single Cb plus a list of Cfs, Takeda and Doi propose two lists, a Center List (entities in a sentence that have become zero or explicit pronouns) and a Possible Center List (entities in a sentence that were overtly expressed). In this treatment, there can thus be more than one Center in the Center List, and any Center (in their terms, any entity which has already been expressed as a reduced form) ranks higher than a mere Possible Center (any entity which has not been expressed by a reduced form). However, some Centers rank higher than others, just as, in the older theories, some Possible Centers (Cfs) rank higher than others.

Their experimental implementation is described in helpful detail. One detail is of special interest. Although their basic mechanism can see antecedents only in the previous utterance, they avoid the need for a global focusing stack [Grosz and Sidner 1986]. Antecedents which are further back are handled in an ad hoc, but apparently effective, manner, using two additional lists: the Past Center List (entities that have previously been a zero or explicit pronoun but do not appear in the current utterance) and the Noun List (entities that have never been a zero or explicit pronoun). Contents of these lists are limited to entities in the three previous utterances.

Interpretation of the current sentence proceeds as follows: Each entity in the four lists receives a salience score. Respecting constraints on co-reference and semantic type, find all possible combinations of antecedents from the four lists. Since each antecedent has a score, the total score for a combination can be a simple sum of the antecedent scores (plus bonus scores according to Kameyama-style "property-sharing" constraints). Choose the best scoring interpretation, update the four lists, and move on.

3 Basic Programs for Centering in ASURA

We now describe the programs and use of CNTR.

3.1 Preparations for Transfer Load

As CNTR is build to run within the ASURA transfer system, the first task is to bring that system up. As the programs are not presently under active development, this can be difficult. Below is the very specific procedure used to enable transfer load on as24 on 94/12/06. It should of course be adapted for other machines.

1. Copy as26:/DB/oldproject/asura/develop/translation/transfer/* to as24.
   It proved desirable to copy files because of problems with paths in compiled files.
2. Make customized files
   as24:/usr/project/asura/develop/translation/transfer/load-basic.lisp.941206.1345
   as24:/usr/project/asura/develop/translation/transfer/rws-v2/load-dtm-mset-index.lisp.941206.1347
   Listings of these files are below.
3. In file ...

... transfer/rws-vs/compiler/hostname_tbl.h
enter line
"as24", "/usr/project/asura/develop/translation/transfer/rws-v2/compiler/rws2.0",
and increment _HOSTNUMBER if necessary

4. In directory
   as24:/usr/project/asura/develop/translation/transfer/rws-v2/compiler/
   at Unix prompt, do
   MAKE CLEAN
5. In the same directory, do
   MAKE SCL
   SCL stands for Sun Common Lisp.
6. In the same directory, do
   MAKE
7. In Lucid Common Lisp 4.0
   (load "/usr/project/asura/develop/translation/transfer/load-basic.lisp.941206.1345")
   during load, answer C (compile) to all prompts

Following are listings of the customized load files mentioned above.

```
**********************************
;;;as24@[/usr/project/asura/develop/translation/transfer/load-basic.lisp.941206.1345
**********************************

;;;load-basic.lisp
;;;941205
;;created by ms by hand to match earlier hard copy
;================================================================
;
; Loading Transfer System
;
;================================================================

(in-package 'user)

(defvar *default-home-directory* "/usr/project/asura/develop/translation/")
(defvar user::*demo_home_directory* "/usr/project/asura/develop/translation/")

(setq *diana-exec-flg* nil)
(setq *use-diana-p* nil)

;
; sentence translation system
;
;
(load (merge-pathnames "transfer/rws-v2/load-dtm-mset-index"
       *default-home-directory*))
```

```
;;;end


**********************************
as24@[/usr/project/asura/develop/translation/transfer/rws-v2/load-dtm-mset-index.lisp.941206.13
**********************************
;;;941205
;;;load-dtm-mset-index.lisp
;;;changed by ms by hand to match hard copy


;;;  mode -*- common-lisp  base: 10
;;;

(in-package 'rws)

(when (boundp 'USER::*DEMO_HOME_DIRECTORY*)
  (setf *RWS-TRANS-DIRECTORY*
(concatenate 'string USER::*DEMO_HOME_DIRECTORY* "transfer/")))
(when (boundp 'USER::*DEFAULT-HOME-DIRECTORY*)
  (setf *RWS-TRANS-DIRECTORY*
    (concatenate 'string USER::*DEFAULT-HOME-DIRECTORY* "transfer/")))
(when (and (boundp '*RWS-TRANS-DIRECTORY*)
   (not (boundp '*RWS-HOME-DIRECTORY*)))
  (setf *RWS-HOME-DIRECTORY*
    (concatenate 'string *RWS-TRANS-DIRECTORY* "rws-v2/")))
;
;
(import 'lucid::define-c-function)
(defvar *HOST_LIST* nil)
(defvar *HOST_as24*
  (cons 'as24 "/usr/project/asura/develop/translation/transfer/")) ;<<<<<<<<<<<<<<<<941205 was
(defvar *HOST_as26* (cons 'as26 "/usr2/project/asura/develop/translation/transfer/"))
(defvar *HOST_as12* (cons 'as12 "/mnt/as26/usr2/project/asura/develop/translation/transfer/"))
(defvar *HOST_as20* (cons 'as20 "/mnt/as26/asura/develop/translation/transfer/"))
(defvar *HOST_as28* (cons 'as28 "/mnt/as26/usr2/project/asura/develop/translation/transfer/"))

(setq *HOST_LIST*
      (list *HOST_as26* *HOST_as12*
    *HOST_as20* *HOST_as24* *HOST_as28*))

;;;    define C function
(define-c-function (c_gethostname "_c_gethostname") (buf)
:result-type :string)
;;;    loading C function
(setf c-object "libc.o")
(when (boundp '*RWS-HOME-DIRECTORY*)
  (setf c-object (concatenate 'string *RWS-HOME-DIRECTORY* "libc.o")))
(lucid::load-foreign-files (list c-object))
;;; LISP function Calling compiler!
(defun real-module-name()
(setq pathname "                                ")
(setq hostname (c_gethostname pathname))
       (let ((s-hostname (read-from-string hostname)))
```

```
(print s-hostname)
  (if (assoc s-hostname *HOST_list*)
      (cdr (assoc s-hostname *HOST_LIST*))
    (loop
     (format t "Please Input Path ==> ")
     (terpri)
     (setq s-hostname (read-line))
     (when  (stringp s-hostname)
   (return s-hostname))))))
(print *HOST_LIST*)
(unless (boundp '*RWS-TRANS-DIRECTORY*)
(setf *rws-trans-directory* (real-module-name)))
(unless (boundp '*RWS-HOME-DIRECTORY*)
    (setf *RWS-HOME-DIRECTORY*
(concatenate 'string *rws-trans-directory* "rws-v2/")))
(unless (boundp '*RWS-RULE-DIRECTORY*)
    (setf *RWS-RULE-DIRECTORY*
(concatenate 'string *rws-trans-directory* "rules/grammar-mset/"))) ;  "transfer/rules/dtm9308/'
(unless (boundp '*RWS-RULE-FILE*)
    (setq *RWS-RULE-FILE*
(concatenate 'string *rws-trans-directory* "rules/grammar-mset/rws.data"))) ;  "transfer/rules/(
;;(cd *RWS-HOME-DIRECTORY*) 92.11.26
(load (merge-pathnames "engine/load.lisp" *RWS-HOME-DIRECTORY*))
(load (merge-pathnames "engine/rws-id" *RWS-HOME-DIRECTORY*))
;(load "type/thesaurus")
(load (merge-pathnames "type/load" *RWS-HOME-DIRECTORY*))
(load (merge-pathnames "compiler/load-v2.lisp" *RWS-HOME-DIRECTORY*))
;;(in-package 'rws)
(load (merge-pathnames "context/context-load.lisp" *rws-home-directory*))
;;(load "rws-v2/tolls/help")
(in-package 'user)
(load (merge-pathnames "rws.index" RWS::*RWS-RULE-DIRECTORY*))
```

3.2 Transfer Load and Transfer Invocation

Once the above preparations are complete, load of the Lisp file TRANSFERFNS.LISP (Appendix 2) should proceeded without problems. This file performs the actual transfer system load, and also includes several useful programs for running and debugging transfer rules. Especially important for present purposes is the function DT ("do transfer"). The head of its definition is shown below.

```
(defun dt (number &key
(debug-transfer nil)
(debug-centering nil)
(initialize-centering nil)
(interactive-centering nil))
...)
```

NUMBER is the index to an analysis output ("anout") feature structure (FS) which must be transfered from its original Japanese form to an English or German form suitable for generation input. The 262 anouts of the MSET, as used in the first major CSTAR demo, are loaded into Lisp during the load of TRANSFERFNS.LISP. The simple program call

```
(dt 1)
```

will for instance invoke transfer for the first utterance of the first dialogue, transforming an anout containing もしもし to an appropriate English structure ("transout") containing HELLO.

The keyword arguments to DT enable transfer debugging; but more important for our purposes now are the last three keywords, which affect centering: DEBUG-CENTERING, INITIALIZE-CENTERING, and INTERACTIVE-CENTERING. The first enable several printouts useful for understanding the operation of the centering programs. The last two will be explained below.

3.3 Transfer Rules Modified for CNTR

CNTR is designed to provide a more general alternative for a number of relatively ad hoc rules which have been used in the past in ASURA to resolve zero pronouns. For instance, rule elps038 is specialized to resolve zero arguments for the Japanese verb 送る only. The rule is listed as an example in Appendix 2, and repeated here for convenience.

```
;;;For DISPLAY ONLY!! During demo running Da-06 and Da-10.
;;;AD HOC RULE for setting RECP to HEARER for translation of Da10.
;;;"Well then, I'll send YOU a registration form."
;;;"Also gut, ich werde IHNEN ein Anmeldeformular schicken."
;;;Made unnecessary by Centering programs. For demo, commented out
;;;-- not used, because not needed.
;
;;[IR]*****************************************************
;;[IR];;; M-SET No.10 (da-10) それでは、登録用紙をお送り致します。
;;[IR];;; M-SET No.50 (d1-13) それでは、登録用紙をお送り致します。
;;[IR];;; M-SET No.132 (d5-16) 後日プログラムと予稿集をお送り致します。
;;[IR];;; M-SET No.206 (d8-24) では早速お送り致します。
;;[IR]*****************************************************
;(rws:defrwschema2 elps038 V RE
;"on <obje reln> 送る -1 IN :MOOD :DECLARATIVE
;   \"動作動詞の現在形の平叙文の主語 --> 一人称\"
;   in= [[reln UNKNOWN-IFT]
;        [agen ?agen]
;        [recp ?recp]
;        [obje [[reln 送る -1] ;;;?action]
;              [aspt unrl]
;              ?rest]]]
;
;;   if input.obje.agen =? [] and ( ?action =? $送る -1 or ?action =? $同封する -1 )
;   if input.obje.agen =? []   ;;;and  ?action =? $送る -1
;   then
;     input.obje.agen = input.agen
;   endif
;
;;   if input.obje.recp =? [] and ( ?action =? $送る -1 or ?action =? $同封する -1 )
;   if input.obje.recp =? []   ;;;and  ?action =? $送る -1
;   then
;     input.obje.recp = input.recp
;   endif
;
```

```
;    RETURN INPUT
;end")
;
```

For CNTR experiments, a version of the ASURA transfer rule set (all.euc) was prepared in which most such rules have been commented out.

In compensation, the rules which handle verbs are to be altered in order to invoke CNTR during their operation (that is, after they are matched but before they complete their rewriting operation). Here are the two rules which were altered in this way for demo purposes. The old versions of the rules must be explicitly removed from virtual memory.[1] We show the old rules and the system function calls which remove them. (Again, we repeat for convenience material which is included in Appendix 2.)

```
(rws::remove-rw-rule2 'defv309)
;;;940406 rule added for demo
;;;Commenting out centering rule for cannonical mset load.
(rws:defrwschema2 center002 t t
  "on <reln> 持つ-1 in :PHASE :J-E :Type :Default
      in= [[reln 持つ-1] ;<<
           [aspt ?aspt] ;<<added constraint
           [agen ?agen]
           [obje ?obje]
           ?rest]
%  (rws::resolve-zero-pronouns-by-centering pairlis)

      out= [[reln have-V-1] ;<<
            [aspt ?aspt] ;<<added constraint
            [agen ?agen]
            [obje ?obje]
            ?rest]

   end"
  )

;;;Comment out or remove following transfer rule for demo use of sentence Da-06 (no.6)
;;;replaced by center002
;(rws:defrwschema2 defv309 V もつ
;"on <reln> 持つ-1 in :PHASE :J-E :Type :Default
;   in= [[reln 持つ-1]
;        [AGEN ?AGEN]
;        [OBJE ?OBJE]
;        ?rest]
;   out= [[reln have-V-1]
;         [AGEN ?AGEN]
;         [OBJE ?OBJE]
;         ?rest]
;end")

;;; - - - - - - - - - - - - - -
```

[1] Beware! Due to an apparent system bug, rule removals must be executed *before* entering the definitions for replacement rules. In reverse order, the replacement rules seem to be removed as well as a side effect.

```
(rws::remove-rw-rule2 'defv046)

(rws:defrwschema2 center003 t t
"on <reln> 送る -1 in :PHASE :J-E :Type :Default
   in= [[reln 送る -1]
        [AGEN ?AGEN]
        [RECP ?RECP]
        [OBJE ?OBJE]
        ?rest]

%  (rws::resolve-zero-pronouns-by-centering pairlis)

   out= [[reln send-V-1]
         [AGEN ?AGEN]
         [RECP ?RECP]
         [OBJE ?OBJE]
         ?rest]
end")

;;;Comment out or remove following transfer rule for demo use of sentence Da-10 (no.10)
;;;replace by center003
;(rws:defrwschema2 defv046 V おく
;"on <reln> 送る -1 in :PHASE :J-E :Type :Default
;   in= [[reln 送る -1]
;        [AGEN ?AGEN]
;        [RECP ?RECP]
;        [OBJE ?OBJE]
;        ?rest]
;
;;  ;;; (d8-6) まず2百字の要約を3月20日までにこちらまでお送りください。
;;      if input.dest == input.recp then
;;
;;           delete dest from ?input
;;
;;      endif
;
;
;   out= [[reln send-V-1]
;         [AGEN ?AGEN]
;         [RECP ?RECP]
;         [OBJE ?OBJE]
;         ?rest]
;end")
```

3.4 The Top-level CNTR Function

Note a Lisp function call within the two replacement rules, (RWS::RESOLVE-ZERO-PRONOUNS-BY-CENTERING PAIRLIS), enabled through the use of the percent sign. This special character allows an escape to Lisp during rule operation.

This function is CNTR's top-level function. Its only argument is the function parameter MY-PAIRLIS. The value of the transfer system variable PAIRLIS should always be passed in to this CNTR function as shown.

PAIRLIS is a variable defined within the transfer system which contains bindings for various rule variables. It is called PAIRLIS because it is a list of pairs, where each pair is a rule feature and its binding, which is local to the rule. PAIRLIS itself is not a global transfer system variable, but rather is local to the functions which operate during rule use. If its value is not passed as shown, the CNTR programs cannot access it.

3.5 Overview of CNTR's Operation

Once CNTR's top-level program receives a list of pairs via PAIRLIS, it takes the following steps:

- Within the PAIRLIS pairs, separate zero and non-zero arguments. Zero arguments represent zero pronouns which must be resolved.
- Compute possible interpretations by finding all possible (allowable) combinations of binding for the zeros. Candidate bindings are the entities (feature structures) made available by the *forward-looking-centers* of the previous utterance, Ui-1. (In the special case that there were no zero arguments, only one interpretation is possible.)
- Eliminate any interpretations which violate constraints. At present, CNTR can eliminate coreference violations (see below), but cannot recognize or eliminate type violations.
- Each surviving interpretation will have been tagged with a transition type. Using a hierarchy of transition types as explained in a previous section, select the highest-scoring interpretation. (Or, if the the function DT has been invoked with a non-nil :INTERACTIVE-CENTERING keyword argument, present a menu of possible interpretations to the user for interactive selection.

Permutations The only programs within CNTR which may be difficult to follow are those which find all possible combinations of zero and non-zero arguments to compose possible "bare interpretations", i.e. potential forward-looking-center sets. The difficulty arises because of the combinatorics: in a way familiar from statistical study, one must find all possible ways of matching a set of candidates against a set of zeros. This is done within the recursive function PERMUTE-ALL-POSSIBLE-INTERPRETATION-SETS. We will not review the computation in detail here, but those interested in altering the process should attend carefully to the debug printouts between "Computation of possible interpretations STARTS" and "Computation of possible interpretations ENDS".

During this stage, potential interpretations in which different features share any bindings are eliminated (III.E.1, filtering coreference violations).

Identifying Centers "Bare interpretations" (forward-looking-center sets) are augmented as CNTR runs to become complete interpretations. Each complete interpretation is a list with four fields (see III.C):

```
transition-type
backward-looking-center
preferred-center
forward-looking-centers
```

The subroutine calls which augment interpretations can be seen in III.F: Calculating a preferred center involves finding the element in the interpretation which scores highest on the syntactic hierarchy *center-salience-ranking*.

Unfortunately, there is a complication: ASURA's analysis output presently delivers features whose names are cases (like agen) rather than true syntactic functions (like subj). Thus it is necessary to intermap these notations – internally, converting case notation to syntactic notation – before sorting. Note a warning in the code: *The present mapping is for demonstration only, and does not address potential problems of ambiguous mappings.*

Calculation of backward-looking-center presents no special problems, but follows from the definition: The backward-looking-center of the current utterance Ui is the most salient entity of the last utterance which appears in Ui, that is, the entity also in Ui which had the most salient role in Ui-1.

Calculation of the transition type and automatic or interactive selection of the highest-ranking interpretation are also straightforward.

Selecting an interpretation Once an interpretation has been selected, it is installed as the updated value of PAIRLIS using the transfer system function rws::install-key-value.

Updating Then the global variables *forward-looking-center-of-last-utterance* and *backward-looking-center-of-last-utterance* are updated, and the operation of CNTR is complete for that rule. (Rather than use these globals, future versions of CNTR could follow Nagata's original concept in using DEFSTRUCT to create structures more suitable for long-range record-keeping.)

3.6 Interaction with Other Resolution Rule in the Demo

As mentioned, CNTR rules run alongside the older rules which implement e.g. Dohsaka's rules using honorifics. In fact, the current demo would not work properly without these rules. The present prototype demo works only for the 2-utterance sequence Da-6, Da-10 (transouts number 6 and 10) – see the function RUN-DEMO (I.B).

In the demo, Da-6 can be treated as an initial utterance because all of its zeros are resolved via honorifics. At present, CNTR treats discourse-initial utterances containing unresolved zeros as errors.

4 Current Limitations and the Future

Two limitations now prevent CNTR from practical operation within the MSET corpus:

First, the program lacks application of type constraints, and thus will allow type violations like "The application form will send me right away" into interpretations. The transfer system does have a typed variable checking facility, but it has been used only for preliminary experiments. Given the power of the facility and its potential usefulness in avoiding many ad hoc aspects of current rules (see [Seligman 1993]), further experimentation might be worthwhile.

Second, the system can resolve zeros at present only if the referents are in the immediately preceding utterance. In the MSET, this often leads to resolution failure, because there are many short utterances like はい which effectively separate zeros from their potential anaphors. Thus some sort of global model is required. However, it need not be elaborate, as [Takeda and Doi 1994] demonstrate.

REFERENCES

Allen, J. 1987. Understanding Natural Language. The Benjamin/Cummings Publishing Company, Inc. Menlo Park, CA, 1987.

Dohsaka, K. 1990. "Identifying the Referents of Zero-pronouns in Japanese Based on Pragmatic Constraint Interpretation," Proc. ECA190, pp. 240-245, 1990.

Hasegawa, T. 1990. Feature Structure Rewriting System Manual (Revised Version). ATR Technical Report TR-I-0187.

Hearst, M. 1994. "Multi-paragraph segmentation of expository text." In Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, June 27-30, 1994.

Kameyama, M. 1985. Zero Anaphora: The Case of Japanese. Ph.D. Dissertation, Stanford University Dept. of Linguistics.

Kameyama, M. 1986. A Property-sahring constraint in centering. In Proc. 24th Annual Meeting of the ACL, Association of Computational Linguistics, pages pp. 200-206, NYC, NY.

Kameyam, M. 1988. Japanese Zero Pronominal Binding, Where Syntax and Sicourse Meet. In Wm. Poser, ed., Papers from the Second International Workshop on Japanese Syntax. Stanford: CSLI. Also available as University of Pennsylvania Tech Report MS-CIS-86-60.

Mitamura, M., E. Nyberg, 3rd, and J. Carbonell. 1993. "Automated Corpus Analysis and the Acquisition of Large, Multi-lingual Knowledge Bases for MT." Proceedings of the Fifth Intl. Conf. on Theoretical and Methodological Issues in Machine Translation. Kyoto, Japan, July 14-16, 1993.

Nasukawa, T. 1994. "Robust Method of Pronoun Resolution Using Full-text Information." In Proceedings of COLING-94, Aug. 5-9, 1994, Kyoto, Japan.

Nomoto, T. and Y. Nitta. 1994. "A Grammatico-statistical Approach to Discourse Partitioning." In Proceedings of COLING-94, Aug. 5-9, 1994, Kyoto, Japan.

Richardson, S., L. Vanderwende, and W. Dolan. 1993. "Combining Dictionary-based and Example-based Methods for Natural Language Analysis." Proceedings of the Fifth Intl. Conf. on Theoretical and Methodological Issues in Machine Translation. Kyoto, Japan, July 14-16, 1993.

Schuetze, H. 1993. Word space. In Advances in Neural Information Processing Systems 5, ed. by Stephen J. Hanson et al, San Mateo, CA: Morgan Kaufmann.

Seligman, M. 1993. A Japanese-German Transfer Component for ASURA. ATR Technical Report TR-I-0365.

Skorochod'ko, E. Adaptive Method of Automatic Abstracting and Indexing. In Information Processing 71: Proceedings of the IFIP Congress 71, ed. by C.V. Freiman, 1179-1182. North-Holland Publishing Co.

Takeda, S. and N. Doi. 1994. Centering In Japanese: A Step Towares Better Inpterpretation of Pronouns and Zero-Pronouns. In Proceedings of COLING-94, Aug. 5-9, 1994, Kyoto, Japan.

Yamashita, Y., H. Yoshida, T. Hiramatsu, Y. Nomura, and R. Mizoguchi. 1993. "MASCOTS II: A Dialog Manager in General Interface for Speech Input and Output". IEICE Trans. Inf. and Syst., Vol E76-D, No. 1 January, 1993.

Walker, M., I. Masayo, and S. Cote. 1992. Japanese Discourse and the Process of Centering. University Of Pennsylvania Tecnical Report, IRCS Report 92-14.

APPENDIX 1: CNTR.LISP Program Code

APPENDIX 2: TRANSFERFNS.LISP Program Code

LINE # | TEXT

```lisp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;==================================================================================
;;;          CNTR: Basic Functions for Centering Experiments with ASURA
;;;==================================================================================
;;;
;;;                               Mark Seligman
;;;                                Dec. 1994
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;CONTENTS

;;;I. AUX
;;;  A. PRINTING
;;;  B. HACKS FOR DEBUGGING

;;;II. TRANSFER RULES

;;;III. CNTR MAIN FNS

;;;  A. TOP LEVEL CONTROL
;;;     1. globals, switches
;;;     2. resolving zero pronouns by centering
;;;  B. GETTING ZERO AND NON-ZERO ARGUMENTS
;;;  C. DEFINING INTERPRETATION DATA STRUCTURE
;;;  D. MAKING A UNIQUE INTERPRETATION
;;;  E. COMPUTING POSSIBLE INTERPRETATIONS
;;;     1. filtering coreference violations
;;;  F. AUGMENTING INTERPRETATIONS
;;;     1. calculating preferred center
;;;     2. calculating backward-looking center
;;;     3. calculating transition type
;;;  G. SELECTING THE INTERPRETATION WITH THE HIGHEST-RANKING TRANSITION TYPE
;;;     1. automatic selection
;;;     2. automatic selection
;;;  H. UPDATING PAIRLIS
;;;  I. UPDATING *FORWARD-LOOKING-CENTERS-OF-LAST-UTTERANCE*
;;;  J. UPDATING *BACKWARD-LOOKING-CENTER-OF-LAST-UTTERANCE*

;;;**********************************************************************************
;;;
;;;  I. AUX
;;;
;;;**********************************************************************************

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  I.A. PRINTING
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setq *print-level* 20)

(defun rws::pprint-all-pairs (list-of-pairs)
  (let ((counter 1))
    (format t "~2%:::::::::::::::::::::::::::::[start list-of-pairs]")
    (loop for pair in list-of-pairs do
          ;;(format t "~2%pair#~s" counter)
          (rws::pprint-pair pair)
          (format t "~%-------------------------")
          ;;(setq counter (1+ counter))
          )
    (format t "~%:::::::::::::::::::::::::::::[end list-of-pairs]~%")))

(defun rws::pprint-pair (pair)
  (format t "~%(~s . ~%" (first pair))
  (rws::ppfs (rest pair))
  (format t "~%)"))

(defun rws::pprint-all-interpretations (list-of-interpretations)
  (let ((counter 1))
    (rws::print-transition-definitions)
    (format t "~2%@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@[start list-of-interpretations]")
    (loop for interpretation in list-of-interpretations do
          (format t "~2%INTERPRETATION#~s" counter)
          (rws::pprint-interpretation interpretation)
          (format t "~%- - - - - - - - - - - - - - - - - - - - - - ")
          (setq counter (1+ counter)))
    (format t "~%@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@[end list-of-interpretations]~%")))

(defun rws::print-transition-definitions ()
  (format t "~2%TRANSITION DEFINITIONS in preference order.")
  (format t "~%Cb is backward-looking-center; Cp is preferred-center; Ui is current utterance. Ui-1 is preceding utterance.")
  (format t "~%Cb(Ui-1) = Cb(Ui) and Cb(Ui) = Cp(Ui): CONTINUE.")
  (format t "~%Cb(Ui-1) = Cb(Ui) and Cb(Ui) not= Cp(Ui): RETAIN.")
  (format t "~%Cb(Ui-1) not= Cb(Ui) and Cb(Ui) = Cp(Ui): SMOOTH-SHIFT.")
  (format t "~%Cb(Ui-1) not= Cb(Ui) and Cb(Ui) not= Cp(Ui): SHIFT."))

(defun rws::pprint-interpretation (interpretation)
  (let ((transition-type
          (rws::get-interpretation-element interpretation 'transition-type))
         (backward-looking-center
          (rws::get-interpretation-element interpretation 'backward-looking-center))
         (preferred-center
          (rws::get-interpretation-element interpretation 'preferred-center))
         (forward-looking-centers
          (rws::get-interpretation-element interpretation 'forward-looking-centers)))

    (format t "~2%Transition type: ~S" transition-type)
    (format t "~2%Backward-looking-center:~%")
    (rws::ppfs backward-looking-center)
    (format t "~2%Preferred-center:~%")
    (rws::ppfs preferred-center)
    (format t "~2%Forward-looking-centers: ~%")
    (rws::pprint-all-pairs forward-looking-centers)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  I.B. HACKS FOR DEBUGGING AND DEMO
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;Not called by functions below.
(defun rws::reset-fcs ()
  (setq *forward-looking-centers-of-last-utterance* nil))

(defun rws::ppfs (node) (rws::pprint-fs node))

(defun rws::run-demo ()
```

```
LINE #  TEXT
   (dt 6 :initialize-centering t :debug-centering t)
   (dt 10 :initialize-centering nil :debug-centering t))

(defun rws::run-demo-interactive ()
  (dt 6 :initialize-centering t :debug-centering t)
  (dt 10 :initialize-centering nil :debug-centering t :interactive-centering t))

;;;*****************************************************************************************
;;;
;;;  II. TRANSFER RULES
;;;
;;;*****************************************************************************************

;;;<<<<<TRANSFER SYSTEM BUG??
;;;Rule removals must be executed BEFORE entering
;;;the definitions for replacement rules.
;;;In reverse order, the replacement rules seem to be
;;;removed as well as a side effect.
(rws::remove-rw-rule2 'defv309)

;;;940406 rule added for demo
;;;Commenting out centering rule for cannonical mset load.
(rws:defrwschema2 center002 t t
  "on <reln> 持つ-1 in :PHASE :J-E :Type :Default
      in= [[reln 持つ-1] ;<<
           [aspt ?aspt] ;<<added constraint
           [agen ?agen]
           [obje ?obje]
           ?rest]
%  (rws::resolve-zero-pronouns-by-centering pairlis)

      out= [[reln have-V-1] ;<<
            [aspt ?aspt] ;<<added constraint
            [agen ?agen]
            [obje ?obje]
            ?rest]

   end"
  )

;;;Comment out or remove following transfer rule for demo use of sentence Da-06 (no.6)
;;;replaced by center002
;(rws:defrwschema2 defv309 V もつ
;"on <reln> 持つ-1 in :PHASE :J-E :Type :Default
;   in= [[reln 持つ-1]
;        [AGEN ?AGEN]
;        [OBJE ?OBJE]
;        ?rest]
;   out= [[reln have-V-1]
;         [AGEN ?AGEN]
;         [OBJE ?OBJE]
;         ?rest]
;end")

;;; - - - - - - - - - - - - - -

(rws::remove-rw-rule2 'defv046)

(rws:defrwschema2 center003 t t
"on <reln> 送る-1 in :PHASE :J-E :Type :Default
   in= [[reln 送る-1]
        [AGEN ?AGEN]
        [RECP ?RECP]
        [OBJE ?OBJE]
        ?rest]

%  (rws::resolve-zero-pronouns-by-centering pairlis)

   out= [[reln send-V-1]
         [AGEN ?AGEN]
         [RECP ?RECP]
         [OBJE ?OBJE]
         ?rest]
end")

;;;Comment out or remove following transfer rule for demo use of sentence Da-10 (no.10)
;;;replace by center003
;(rws:defrwschema2 defv046 V おく
;"on <reln> 送る-1 in :PHASE :J-E :Type :Default
;   in= [[reln 送る-1]
;        [AGEN ?AGEN]
;        [RECP ?RECP]
;        [OBJE ?OBJE]
;        ?rest]
;
;  ;;; (d8-6)まず2百字の要約を3月20日までにこちらまでお送りください。
;      if input.dest == input.recp then
;
;          delete dest from ?input
;
;      endif
;
;
;   out= [[reln send-V-1]
;         [AGEN ?AGEN]
;         [RECP ?RECP]
;         [OBJE ?OBJE]
;         ?rest]
;end")

;;; - - - - - - - - - - - - - -

;;;For DISPLAY ONLY!! During demo running Da-06 and Da-10.
;;;AD HOC RULE for setting RECP to HEARER for translation of Da10.
;;;"Well then, I'll send YOU a registration form."
;;;"Also gut, ich werde IHNEN ein Anmeldeformular schicken."
;;;Made unnecessary by Centering programs. For demo, commented out
;;;-- not used, because not needed.
;
;;[IR]*****************************************************
;;[IR];;; M-SET No.10 (da-10) それでは、登録用紙をお送り致します。
;;[IR];;; M-SET No.50 (d1-13) それでは、登録用紙をお送り致します。
;;[IR];;; M-SET No.132 (d5-16) 後日プログラムと予稿集をお送り致します。
;;[IR];;; M-SET No.206 (d8-24) では早速お送り致します。
;;[IR]*****************************************************
;(rws:defrwschema2 elps038 V RE
;"on <obje reln> 送る-1 IN :MOOD :DECLARATIVE
;   \"動作動詞の現在形の平叙文の主語 --> 一人称\"
;   in= [[reln UNKNOWN-IFT]
;        [agen ?agen]
```

LINE # TEXT

```
;        [recp ?recp]
;        [obje [[reln 送る-1] ;;;?action]
;              [aspt unrl]
;              ?rest]]]
;
;;   if input.obje.agen =? [] and ( ?action =? $送る-1 or ?action =? $同封する-1 )
;   if input.obje.agen =? []   ;;;and  ?action =? $送る-1
;   then
;     input.obje.agen = input.agen
;   endif
;
;;   if input.obje.recp =? [] and ( ?action =? $送る-1 or ?action =? $同封する-1 )
;   if input.obje.recp =? []   ;;;and  ?action =? $送る-1
;   then
;     input.obje.recp = input.recp
;   endif
;
;   RETURN INPUT
;end")
;

;;;*****************************************************************************************
;;;
;;;  III. CNTR MAIN FUNCTIONS
;;;
;;;*****************************************************************************************

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  III.A. TOP LEVEL CONTROL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;=============================================================================================
;;;  III.A.1.  GLOBALS, SWITCHES
;;;=============================================================================================

(defvar *forward-looking-centers-of-last-utterance* nil)

(defvar *backward-looking-center-of-last-utterance* nil)

(defvar *interactive-centering* nil)

(defvar *debug-centering* nil)

(defvar *initialize-centering* nil)

(defun rws::initialize-on ()
  (setq *initialize-centering* t))

(defun rws::initialize-off ()
  (setq *initialize-centering* nil))

(defun rws::interactive-centering-on ()
  (setq *interactive-centering* t))

(defun rws::interactive-centering-off ()
  (setq *interactive-centering* nil))

(defun rws::debug-centering-on ()
  (setq *debug-centering* t))

(defun rws::debug-centering-off ()
  (setq *debug-centering* nil))

;;;=============================================================================================
;;;  III.A.2.  RESOLVING ZERO PRONOUNS BY CENTERING
;;;=============================================================================================

;;;PAIRLIS is not global, and scoping is lexical,
;;;so this function does not see PAIRLIS if not passed in.
(defun rws::resolve-zero-pronouns-by-centering (mypairlis)
  ;;(A) get all zero-arg-pairs and non-zero-arg-pairs for current utterance
  (let* ((zero-results (rws::get-zero-and-non-zero-arg-pairs mypairlis))
         (zero-arg-pairs (first zero-results))
         (non-zero-arg-pairs (second zero-results)))

    (if *initialize-centering*
        (progn
          (if zero-arg-pairs
              (warning "Found ZERO ARGUMENT in DISCOURSE-INITIAL UTTERANCE.~%Centering algorithms may be distorted."))
          (format t "~%INITIALIZE-CENTERING: SETTING *FORWARD-LOOKING-CENTERS-OF-LAST-UTTERANCE* TO NIL.")
          (setq *forward-looking-centers-of-last-utterance* nil)
          (format t "~%INITIALIZE-CENTERING: SETTING *BACKWARD-LOOKING-CENTER-OF-LAST-UTTERANCE* TO NIL.")
          (setq *backward-looking-center-of-last-utterance* nil)))

    (if *debug-centering*
        (progn
          (format t "~2%mypairlis: ~S" mypairlis)

          (format t "~2%*forward-looking-centers-of-last-utterance*:")
          (rws::pprint-all-pairs *forward-looking-centers-of-last-utterance*)

          ;;if last utterance was the first in a discourse, there will be no backward-looking center
          (format t "~2%*backward-looking-center-of-last-utterance*:~%")
          (rws::ppfs *backward-looking-center-of-last-utterance*)

          (format t "~2%zero-arg-pairs of current utterance: ~S~%" zero-arg-pairs)))

    (if *initialize-centering*
        ;;if initialize-centering, this is the first utterance in a dialogue segment
        (let* ((unique-interpretation
                 (rws::make-unique-interpretation mypairlis))
               (forward-looking-centers-for-current-utterance
                 (rws::get-interpretation-element unique-interpretation 'forward-looking-centers)))

          (if *debug-centering*
              (progn
                (format t "~%********UNIQUE INTERPRETATION OF THIS INITIAL UTTERANCE:********~%")
                (rws::pprint-interpretation unique-interpretation)))

          ;;no need to modify pairlis

          ;;(A) set new *forward-looking-centers-of-last-utterance* to new, completed pairlis
          ;;the current utterance has no backward-looking-center since it is the first,
          ;;so *backward-center-of-last-utterance* will not be updated now
          (rws::update-forward-looking-centers unique-interpretation)

          ;;<<<nullify *intialize-centering* so that next utterance will not
          ;;be treated as dialogue-initial
          (rws::initialize-off))
```

LINE # TEXT

```
361         ;;Else this is not the first utterance.
362         ;;Compute all possible interpretations and
363         ;;choose the interpretation which scores highest on transition hierarchy.
364         (let* ((possible-interpretations
365                  (rws::compute-possible-interpretations
366                   zero-arg-pairs non-zero-arg-pairs))
367                (highest-ranking-interpretation
368                 (if *interactive-centering*
369                     (rws::get-highest-ranking-interpretation-from-user
370                      possible-interpretations)
371                     (rws::get-highest-ranking-interpretation
372                      possible-interpretations)))
373                (forward-looking-centers-for-current-utterance
374                 (rws::get-interpretation-element highest-ranking-interpretation 'forward-looking-centers))
375                (backward-looking-center-for-current-utterance
376                 (rws::get-interpretation-element highest-ranking-interpretation 'backward-looking-center)))
377
378           ;;(A) install values from best interpretation into mypairlis
379           ;;<<<<beware scoping
380           (rws::install-interpretation-in-pairlis highest-ranking-interpretation mypairlis)
381
382           (if *debug-centering*
383               (format t "~2%mypairlis after installation: ~S" mypairlis))
384
385           ;;(A) set new *forward-looking-centers-of-last-utterance* to new, completed pairlis
386           ;;(A) set new *backward-center-of-last-utterance*
387           ;;(rws::update-centers mypairlis)
388           (rws::update-forward-looking-centers highest-ranking-interpretation)
389           (rws::update-backward-looking-center highest-ranking-interpretation))))
390
391   (if *debug-centering*
392       (progn
393         (format t "~2%*forward-looking-centers-of-last-utterance* after update:")
394         (rws::pprint-all-pairs *forward-looking-centers-of-last-utterance*)
395         (format t "~2%*backward-looking-center-of-last-utterance* after update:")
396         (rws::ppfs *backward-looking-center-of-last-utterance*))))
397
398 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
399 ;;;   III.B. GETTING ZERO AND NON-ZERO ARGUMENTS
400 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
401
402 ;;;Returns (<zero-arg-pairs><non-zero-arg-pairs>)
403 (defun rws::get-zero-and-non-zero-arg-pairs (mypairlis)
404   (let* ((zero-arg-pairs
405           (loop for pair in mypairlis if (rws::pair-zero-argp pair)
406                 collect pair))
407          ;;<<<note filtering
408          (filtered-zero-arg-pairs
409           (rws::filter-args-of-interest zero-arg-pairs))
410          (non-zero-arg-pairs
411           (set-difference mypairlis zero-arg-pairs))
412          ;;<<<note filtering
413          (filtered-non-zero-arg-pairs
414           (rws::filter-args-of-interest non-zero-arg-pairs)))
415     (list filtered-zero-arg-pairs filtered-non-zero-arg-pairs)))
416
417 (defvar *args-of-interest* '(?agen ?objе ?recp))
418
419 ;;;For both zero and non-zero args, we consider only
420 ;;;members of *ARGS-OF-INTEREST* as defined above.
421 ;;;Otherwise ?REST, ?ROOT, ?INPUT, and other features of PAIRLIS
422 ;;;interfere in centering calculations.
423 (defun rws::filter-args-of-interest (list-of-pairs)
424   (loop for pair in list-of-pairs
425         when (member (car pair) *args-of-interest*)
426         collect pair))
427
428 (defun rws::pair-zero-argp (pair)
429   ;;(format t "~%pair is: ~S" pair)
430   (let* ((pair-node (rest pair)))
431     (and
432      (rws::node-p pair-node)
433      (rws::null-node-p pair-node))))
434
435 (defun rws::null-node-p (node)
436   (rws::fs-equal node (rws::read-fs-from-string "[]")))
437
438 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
439 ;;;  III.C. DEFINING INTERPRETATION DATA STRUCTURE
440 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
441
442 (defvar *interpretation-elements*
443   '((transition-type 0)
444     (backward-looking-center 1)
445     (preferred-center 2)
446     (forward-looking-centers 3)))
447
448 (defun rws::get-interpretation-element-position (element-name)
449   (second
450    (assoc element-name *interpretation-elements*)))
451
452 (defun rws::get-interpretation-element (interpretation element-name)
453   (let ((position (rws::get-interpretation-element-position element-name)))
454     (nth position interpretation)))
455
456 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
457 ;;;  III.D. MAKING A UNIQUE INTERPRETATION
458 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
459
460 (defun rws::make-unique-interpretation (mypairlis)
461   (let* ((forward-looking-centers
462           (rws::filter-args-of-interest mypairlis))
463          (preferred-center
464           (rws::get-interpretation-preferred-center forward-looking-centers)))
465
466     ;;MAKING UNIQUE INTERPRETATION
467     ;;transition type is NO-TRANSITION, backward-looking-center is null
468     (no-transition nil ,preferred-center ,forward-looking-centers)))
469
470 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
471 ;;;  III.E. COMPUTING POSSIBLE INTERPRETATIONS
472 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
473
474 ;;;Pair format: (?case-tag . <node ftr str>)
475 ;;;Both zero-arg-pairs and *forward-looking-centers-of-last-utterance*
476 ;;;have form (<ctr><ctr> ...)
477 (defun rws::compute-possible-interpretations (zero-arg-pairs non-zero-arg-pairs)
478   (let* ((all-partial-interpretation-sets
479           (rws::get-all-partial-interpretation-sets zero-arg-pairs))
480          (interpretations-without-non-zero-args
```

LINE # | TEXT

```
          (rws::permute-all-partial-interpretation-sets all-partial-interpretation-sets))
         ;;filtering for coreference violations after adding non-zero args
         ;;sorting is suppressed
         (once-filtered-interpretations-without-non-zero-args
          (rws::filter-coreference-violations-and-sort interpretations-without-non-zero-args))
         (once-filtered-interpretations-plus-non-zero-args
          (loop for interpretation in once-filtered-interpretations-without-non-zero-args collect
                (append non-zero-arg-pairs interpretation)))
         ;;re-filtering for coreference violations after adding non-zero args
         ;;this time, sort results by salience
         (refiltered-interpretations-sorted
           (rws::filter-coreference-violations-and-sort once-filtered-interpretations-plus-non-zero-args :sort t))
         (augmented-refiltered-interpretations
          (loop for interpretation in refiltered-interpretations-sorted collect
                (rws::augment-interpretation interpretation))))
    (if *debug-centering*
        (progn
          (format t "~2%;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;")
          (format t "~%Computation of possible interpretations STARTS.")
          (format t "~%;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;")
          ;;(format t "~2%all-partial-interpretation-sets:")
          (format t "~2%interpretations-without-non-zero-args:")
          (loop for interpretation in interpretations-without-non-zero-args do
                (rws::pprint-all-pairs interpretation))
          (format t "~2%once-filtered-interpretations-without-non-zero-args:")
          (loop for interpretation in once-filtered-interpretations-without-non-zero-args do
                (rws::pprint-all-pairs interpretation))
          (format t "~2%once-filtered-interpretations-plus-non-zero-args:")
          (loop for interpretation in once-filtered-interpretations-plus-non-zero-args do
                (rws::pprint-all-pairs interpretation))
          (format t "~2%refiltered-interpretations-sorted:")
          (loop for interpretation in refiltered-interpretations-sorted do
                (rws::pprint-all-pairs interpretation))
          (format t "~2%;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;")
          (format t "~%Computation of possible interpretations ENDS.")
          (format t "~%;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;")

          (format t "~2%********FOR THIS NON-INITIAL UTTERANCE, POSSIBLE INTERPRETATIONS ARE:******~%")
          (rws::pprint-all-interpretations augmented-refiltered-interpretations)))

    augmented-refiltered-interpretations))

;;;ZERO-ARG-PAIRS is list of pairs e.g.
;;;((?agt . [])(?obj . [])(?recp . [])) --- for an utterance with three zero args
;;; and *forward-looking-centers-of-last-utterance* is similar but with non-zero-bindings
;;;((?agt . [fs1])(?obj . [fs2])(?recp . [fs3]) ...) giving a list of entities
;;;(feature structures) [fs1][fs2][fs3] ...
;;;This function returns all Partial Interpretation Sets, where each PIS is a list of all
;;;possible pairings of one case (or syntactic) role against all entities appearing in
;;;the previous utterance Ui-1.
;;;(   (((?agt . [fs1]))  ;partial interpretation 1     ;a partial interpretation set
;;;      ((?agt . [fs2]))  ;partial interpretation 2     ;(cont)
;;;      ((?agt . [fs3]))) ;partial interpretation 3     ;(cont)
;;;     <another partial interpretation set>
;;;     <another partial interpretation set> ...)
(defun rws::get-all-partial-interpretation-sets (zero-arg-pairs)
  (loop for zero-arg-pair in zero-arg-pairs collect
        (loop for forward-looking-center in *forward-looking-centers-of-last-utterance* collect
              (list (rws::bind-zero-arg-pair-and-forward-looking-center
                     zero-arg-pair forward-looking-center)))))

;;;ZERO-ARG-PAIR looks like (?agt . [])
;;;and forward-looking-center pair looks like (?obj . [non-empty ftr str])
;;;Result of merging is e.g. (?agt . [non-empty ftr str]),
;;;where cdr of forward-looking-center replaces empty cdr of zero-arg-pair: (?obj . [non-empty ftr str])
(defun rws::bind-zero-arg-pair-and-forward-looking-center (zero-arg-pair forward-looking-center)
  (cons (car zero-arg-pair)(cdr forward-looking-center)))

;;;Recursively combines partial interpretation sets into all possible
;;;combinations of zero args and entities.
;;;Non-zero args are not yet included.
(defun rws::permute-all-partial-interpretation-sets (partial-interpretation-sets)
  (let ((output nil))
    ;;if only one partial-intrpretation-set (or none) return it only
    (cond ((< (length partial-interpretation-sets) 2 )
           (first partial-interpretation-sets))
          ;;if 2 partial-intrpretation-sets, representing two zero args, e.g.
          ;; ?agen and ?objе, bottom out and permute
          ((equal 2 (length partial-interpretation-sets))
           (let ((first-list (first partial-interpretation-sets))
                 (second-list (second partial-interpretation-sets)))
             (loop for x in first-list do
                   (loop for y in second-list do
                         (setq output (cons (append x y) output))))
             (reverse output)))

          ;;else recurse
          (t (rws::permute-all-partial-interpretation-sets
              (list (first partial-interpretation-sets)
                    (rws::permute-all-partial-interpretation-sets (rest partial-interpretation-sets))))))))

;;;=========================================================================
;;;  III.E.1.  FILTERING COREFERENCE VIOLATIONS
;;;=========================================================================

;;;Input is a set of "bare" interpretations -- not yet complete interpretation data structures --
;;;where each interpretation is a list of pairs
;;;e.g. (    ((?AGEN . #<Structure RWS::NODE E30EEE>) ...)    ((?AGEN . #<Structure RWS::NODE E30E8E>) ...)     )
(defun rws::filter-coreference-violations-and-sort (interpretations &key (sort nil)) ;<<<941207 default is no sort
  (let ((output nil))
    (loop for interpretation in interpretations do
          (let ((entities
                 (loop for pair in interpretation collect
                       (cdr pair))))
            (if (not (rws::fs-list-contains-repetitions? entities))
                (if sort
                    ;;if explicitly requested
                    ;;sort pairs in this possible interpretation by salience
                    (let* ((sorted-pairs-for-interpretation
                            (rws::sort-case-based-pairs interpretation)))
                      (setq output (cons sorted-pairs-for-interpretation output)))
                    ;;if not, return the interpretation as it is
                    (setq output (cons interpretation output))))))
    (reverse output)))

(defun rws::fs-list-contains-repetitions? (list)
  (let ((found? nil))
    (loop for x in list until found? do
          ;;equal does work for feature structures
          (let ((list-minus-x (remove x list :test #'rws::fs-equal)))
```

LINE # TEXT

```
              (if (> (- (length list) (length list-minus-x) ) 1)
                  (setq found? t))))
      found?))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  III.F. AUGMENTING INTERPRETATIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;Input is a "bare" interpretation -- not yet a complete interpretation data structure --
;;;in the form (<pair><pair> ...)
;;;Output is a complete interpretation data structure
;;;of form (transition-type backward-looking-center preferred-center interpretation)
(defun rws::augment-interpretation (interpretation)
  (let* ((preferred-center
           (rws::get-interpretation-preferred-center interpretation))
          (backward-looking-center
           (rws::get-interpretation-backward-looking-center interpretation))
          (transition-type
           (rws::get-interpretation-transition-type backward-looking-center preferred-center)))

    ;;returning augmented, complete, interpretation
    (list transition-type backward-looking-center preferred-center interpretation)))

;;;==========================================================================================
;;;  III.F.1.  CALCULATING PREFERRED CENTER
;;;==========================================================================================

(defun rws::get-interpretation-preferred-center (interpretation)
  (cdr (rws::get-most-salient-pair interpretation)))

;;;CASE-BASED pairs have e.g. ?agen as car.
;;;SYNTAX-BASED have e.g. ?subj.
;;;If input list-of-pairs is case-based, this function
;;;converts to syntax-based internally
;;;to enable sorting by syntax-based salience.
;;;Always delivers case-based output.
(defun rws::get-most-salient-pair (list-of-pairs &key (input-type 'case))
  (if (equal input-type 'case)
      ;;sort-case-based-pairs
      ;;internatlly converts into syntax-based for sorting
      ;;and then back to case-based for output
      (first (rws::sort-case-based-pairs list-of-pairs))
      (first (rws::sort-syntax-based-pairs list-of-pairs))))

;;;Internally converts to syntax-based list of pairs,
;;;sorts according to syntactic salience, and then
;;;reconverts to case-based, preserving ordering.
(defun rws::sort-case-based-pairs (case-based-pairs)
    (let* ((syntax-based-pairs
             (loop for pair in case-based-pairs collect
                   (let* ((case (car pair))
                          (syntactic-function
                           (rws::get-syntactic-function-for-case case)))
                     (cons syntactic-function (cdr pair)))))
           (sorted-syntax-based-pairs
            (rws::sort-syntax-based-pairs syntax-based-pairs))
           (sorted-case-based-pairs
            (loop for pair in sorted-syntax-based-pairs collect
                  (let* ((syntactic-function (car pair))
                         (case
                          (rws::get-case-for-syntactic-function syntactic-function)))
                    (cons case (cdr pair))))))
      sorted-case-based-pairs))

(defun rws::sort-syntax-based-pairs (syntax-based-pairs)
  (sort
   syntax-based-pairs
   #'rws::center-salience-greaterp :key #'first))

;;;Current simple format for a center, or entity: (?case-tag . <node str>)
;;;Thus *forward-looking-centers-of-last-utterance* becomes a subset of pairlis -- same format.
;;;(?agen ?obje ?recp)

(defvar *center-salience-ranking* nil)

;;;Map cases into syntactic functions.
;;;This mapping is incomplete -- for
;;;demonstration only -- and does not address problems of
;;;ambiguous mappings.

;;;Walker et al use this hierarchy: topic :empathy :subj :obj2 :obj
;;;'(?agen ?obje ?recp))
(setq *center-salience-ranking*
      '(subj obj2 obj))

(defvar *case-to-syntactic-function-mapping*
  '((?agen subj)
    (?obje obj)
    (?recp obj2)))

(defvar *syntactic-function-to-case-mapping*
  '((subj ?agen)
    (obj ?obje)
    (obj2 ?recp)))

(defun rws::get-syntactic-function-for-case (case)
  (second (assoc case *case-to-syntactic-function-mapping*
                 :test #'equal)))

(defun rws::get-case-for-syntactic-function (syntactic-function)
  (second (assoc syntactic-function *syntactic-function-to-case-mapping*
                 :test #'equal)))

(defun rws::center-salience-lessp (center1 center2)
  (< (length (member center1 *center-salience-ranking*))
     (length (member center2 *center-salience-ranking*))))

(defun rws::center-salience-greaterp (center1 center2)
  (> (length (member center1 *center-salience-ranking*))
     (length (member center2 *center-salience-ranking*))))

;;;==========================================================================================
;;;  III.F.2.  CALCULATING BACKWARD LOOKING CENTER
;;;==========================================================================================

;;;Input to this function is a "bare" interpretation, i.e. a list of pairs.
;;;Backward-looking-center of the current utterance Ui is the most
;;;salient entity of the last utterance which appears in Ui,
;;;that is, the entity also in Ui which had the most salient role in Ui-1.
;;;Output is this entity, a feature structure.
```

LINE # TEXT

```
721 (defun rws::get-interpretation-backward-looking-center (interpretation)
722   (let* (;;list of feature structures
723          (entities-in-this-interpretation
724           (loop for pair in interpretation collect
725                 (cdr pair)))
726          ;;list of feature structures
727          (entities-in-last-utterance
728           (loop for pair in *forward-looking-centers-of-last-utterance* collect
729                 (cdr pair)))
730          ;;list of feature structures
731          (entities-in-last-utterance-also-in-this-interpretation
732             (intersection entities-in-last-utterance entities-in-this-interpretation
733                           :test #'rws::fs-equal))
734          (pairs-in-last-utterance-also-in-this-interpretation
735           (loop for entity in entities-in-last-utterance-also-in-this-interpretation collect
736                 (rassoc entity *forward-looking-centers-of-last-utterance* :test #'rws::fs-equal)))
737          (most-salient-pair-in-last-utterance-also-in-this-interpretation
738           (rws::get-most-salient-pair pairs-in-last-utterance-also-in-this-interpretation)))
739     (cdr most-salient-pair-in-last-utterance-also-in-this-interpretation)))
740
741 ;;;=============================================================================
742 ;;;  III.F.3.  CALCULATING TRANSITION TYPE
743 ;;;=============================================================================
744
745 ;;;BACKWARD-LOOKING-CENTER and PREFERRED-CENTER are feature structures of the current utterance.
746 ;;;Do not confuse *backward-looking-center-of-last-utterance* with
747 ;;;the BACKWARD-LOOKING-CENTER of the current utterance Ui, which is input to this function.
748 (defun rws::get-interpretation-transition-type (backward-looking-center preferred-center)
749   ;;If this is the first utterance of a dialogue segment, there can be no transition.
750   (if *initialize-centering*
751       'no-transition
752       ;;If this is not the first utterance of a dialogue segment,
753       ;;in the special case of a second utterance, *backward-looking-center-of-last-utterance*
754       ;;will be nil. Note that this is treated as a continuation of backward-looking-center from Ui-1 to Ui.
755       (if (or (null *backward-looking-center-of-last-utterance*)
756               (rws::fs-equal backward-looking-center *backward-looking-center-of-last-utterance*))
757           (if (rws::fs-equal backward-looking-center preferred-center)
758               'continue
759               'retain)
760           (if (rws::fs-equal backward-looking-center preferred-center)
761               'smooth-shift
762               'shift))))
763
764 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
765 ;;;  III.G. SELECTING THE INTERPRETATION WITH THE HIGHEST RANKING TRANSITION TYPE
766 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
767
768 ;;;=============================================================================
769 ;;;  III.G.1.  automatic selection
770 ;;;=============================================================================
771
772 ;;;Each POSSIBLE-INTERPRETATION is of the form
773 ;;;(transition-type backward-looking-center preferred-center bare-interpretation)
774 (defun rws::get-highest-ranking-interpretation (possible-interpretations)
775   (first
776    (sort
777     possible-interpretations
778     #'rws::transition-type-preference-greaterp :key #'first)))
779
780 (defvar *transition-type-ranking* nil
781   "The preferred ranking of transition states")
782
783 (setq *transition-type-ranking*
784       '(continue retain smooth-shift shift no-transition))
785
786 (defun rws::transition-type-preference-lessp (state1 state2)
787   (declare (special *transition-states-ranking*))
788   (assert (and state1 state2))
789   (< (length (member state1 *transition-type-ranking*))
790      (length (member state2 *transition-type-ranking*))))
791
792 (defun rws::transition-type-preference-greaterp (state1 state2)
793   (declare (special *transition-type-ranking*))
794   (assert (and state1 state2))
795   (> (length (member state1 *transition-type-ranking*))
796      (length (member state2 *transition-type-ranking*))))
797
798 ;;;=============================================================================
799 ;;;  III.G.2.  interactive selection
800 ;;;=============================================================================
801
802 (defun rws::get-highest-ranking-interpretation-from-user (possible-interpretations)
803   (let* ((response nil)
804          (selection nil)
805          (sorted-interpretations
806           (sort
807            possible-interpretations
808            #'rws::transition-type-preference-greaterp :key #'first))
809          (number-of-interpretations (length sorted-interpretations)))
810
811     (format t "~2%POSSIBLE INTERPRETATIONS (BELOW) ORDERED BY TRANSITION PREFERENCE:")
812     (rws::pprint-all-interpretations sorted-interpretations)
813     (format t "~2%POSSIBLE INTERPRETATIONS (ABOVE) ORDERED BY TRANSITION PREFERENCE:")
814     (format t "~2%SELECT BY TYPING NUMBER: ")
815     (setq response (read))
816     (loop until (and (integerp response)
817                      (>= response 1)
818                      (<= response number-of-interpretations))
819           do
820           (format t "~%RESPONSE MUST BE INTEGER FROM 1 TO ~s." number-of-interpretations)
821           (format t "~2%Select by typing number: ")
822           (setq response (read)))
823     (nth (1- response) sorted-interpretations)))
824
825 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
826 ;;;   III.H. UPDATING PAIRLIS
827 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
828
829 (defun rws::install-interpretation-in-pairlis (interpretation mypairlis)
830   (let ((forward-looking-centers
831          (rws::get-interpretation-element interpretation 'forward-looking-centers)))
832     (loop for pair in forward-looking-centers do
833           (let ((case (first pair))
834                 (value (rest pair)))
835       (rws::install-key-value case value mypairlis)))))
836
837 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
838 ;;;   III.I. UPDATING *FORWARD-LOOKING-CENTERS-OF-LAST-UTTERANCE*
839 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
840
```

LINE # TEXT

```
(defun rws::update-forward-looking-centers (interpretation)
  (let ((forward-looking-centers
         (rws::get-interpretation-element interpretation 'forward-looking-centers)))
    (setq *forward-looking-centers-of-last-utterance* forward-looking-centers)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;   III.J. UPDATING *BACKWARD-LOOKING-CENTER-OF-LAST-UTTERANCE*
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun rws::update-backward-looking-center (interpretation)
  (let ((backward-looking-center
         (rws::get-interpretation-element interpretation 'backward-looking-center)))
    (setq *backward-looking-center-of-last-utterance* backward-looking-center)))


;;;end of seligman file
```

```
(in-package :user)
;;;loads ruleset for all.921028.euc


;;;*****************************************************************************
;;;LOADING
;;;*****************************************************************************

;;;load transfer system
;;;have to do this before mentioning rws::
(load "/usr/project/asura/develop/translation/transfer/load-basic.lisp")

;;;load printing fns for output
(load "~/aux/readfns.lisp")

;;;---------------------------------------------------------------------------
;;;In load-mset.lisp is a loop, loading
;;;my own (usually changed) files from
;;;"/home/seligman/transfer/rules/grammar-mset/"

;;;7/30/92 substituting a single file containing all rules

(defun load-trules ()
  (rws::remove-all-rw-rules2)
  (load "~/transfer/rules/all.euc")) ;<<<renamed was all.921028.euc

;;;---------------------------------------------------------------------------

(defun load-anouts ()
  (setq fs (rws::push-fs-from-file
            "/usr/project/asura/develop/translation/transfer/mset-demo.fix")))

;;;---------------------------------------------------------------------------

;;;shortcut
(defun load-trules-and-anouts ()
  (load-trules)
  (load-anouts))

(load-trules-and-anouts)

;;;---------------------------------------------------------------------------

;;;alters printout of tranfer results: yields proper "loop" format for generation input
(load "~/transfer/fns/debug-patch.lisp")

;;;assures new version of this file
;;;****until 1/12/93 (load "/home2/nadine91/transfer/rws-v2/engine/transfer1.lisp.9204")
;;;(load "/home/seligman/transfer/rws-v2/engine/transfer1.lisp.9204")

;;;same as above -- avoids extraneous printed material in outdata file
(load "~/transfer/fns/transfer1-patch.lisp")


;;;*****************************************************************************
;;;EXECUTE
;;;*****************************************************************************

;;;do transfer
;;;94/03/03 adding initialize-centering key
;;;940311 adding key args for interactive and debug-centering
;;;beware errors -- some changes made (fixed?) in transrferfns.lisp which should be here instead
(defun dt (number &key (debug-transfer nil) (debug-centering nil)(initialize-centering nil)(interactive-centering nil))
  (if initialize-centering
      (rws::initialize-on)   ;setq *initialize-centering* t
      (rws::initialize-off)) ;setq *initialize-centering* nil

  (if debug-centering
      (rws::debug-centering-on)   ;setq *debug-centering* t
      (rws::debug-centering-off)) ;setq *debug-centering* nil

  (if interactive-centering
      (setq *interactive-centering* t)
      (setq *interactive-centering* nil))

  (if debug-transfer (bug-on)(bug-off))
  (let ((anout (nth (1- number) fs)))
    (rws::trans anout)))

;;;do multi-transfer
(defun dmt (start end)
  (rws::trans-range-exec fs start end nil))

;;;*****************************************************************************
;;;DEBUG and PRINT
;;;*****************************************************************************


(defun debugt ()
  (rws::debug-transfer))

;;;print transfer rule
(defun ptr (rule)
  (rws::print-rw-rule2 rule))

(defun bug-on ()
  ;(setq rws::*debug-transfer-input* t)
  (setq rws::*debug-transfer* t)
  (setq rws::*debug-apply-rw-rule* t)
  (setq rws::*debug-rewrite-subfs* t) ;<<<<nil)
  (setq rws::*debug-fs-match* t)
  (setq rws::*debug-parameter-setting* nil)
  (setq rws::*preference-fig* nil))

(defun bug-off ()
  ;(setq rws::*debug-transfer-input* t)
  (setq rws::*debug-transfer* t)
  (setq rws::*debug-apply-rw-rule* nil)
  (setq rws::*debug-rewrite-subfs* nil)
  (setq rws::*debug-fs-match* nil)
  (setq rws::*debug-parameter-setting* nil)
  (setq rws::*preference-fig* nil))

(defun input-on ()
  (setq rws::*debug-transfer-input* t))

(defun input-off ()
  (setq rws::*debug-transfer-input* nil))

;;;as in debug-patch.lisp -- repeated for easy reference
```

```
LINE #  TEXT
  121   ;(defun loop-on ()
  122   ;  (setq rws::*debug-transfer-loop-format* t))
  123
  124   ;(defun loop-off ()
  125   ;  (setq rws::*debug-transfer-loop-format* nil))
  126
  127
  128   ;;;<<<<<<<<<<careful!!!
  129   (setq rws::*out-pathname* "~/transfer/outdata/")
  130
  131   ;;;as in debug1.lisp -- repeated for easy reference
  132   ;(defun user::open_output (&optional (filename "#dbg_out#"))
  133   ;  (setq rws::*out_fullpath* (merge-pathnames filename rws::*out-pathname*))
  134   ;  (setq rws::*msg_print_p* t))
  135
  136   ;(defun user::close_output ()
  137   ;  (setq rws::*msg_print_p* nil))
  138
  139   ;;;shortcut for making transout files
  140   (defun transouts-on (file)
  141     (input-off)
  142     (loop-on)
  143     (open_output file))
  144
  145   (defun transouts-off ()
  146     (input-on)
  147     (loop-off)
  148     (close_output))
  149
  150   ;;;9/30/92
  151   ;;;revising: y/n numbers? replaced by start-no
  152   (defun mytrans (start end file &optional start-no)
  153     (if (probe-file file)
  154         (delete-file file))
  155     (transouts-on file)
  156
  157     ;;;write unnumbered transouts to output file
  158     (dmt start end)
  159
  160     ;;;add numbers to output file
  161     (if start-no (copy-loop file file start-no))
  162
  163     (transouts-off))
  164
  165
  166
  167   ;;;*****************************************************************************
  168   ;;;REMOVE RULES
  169   ;;;*****************************************************************************
  170
  171   (defun rr (rule)
  172     (rws::remove-rw-rule2 rule))
  173
  174   (defun remove-all-rules ()
  175     (rws::remove-all-rw-rules2))
  176
  177   (defun rulep (rule)
  178      (rws::find-rw-rule2 rule))
  179
  180
  181   ;;;*****************************************************************************
  182   ;;; CENTERING
  183   ;;;*****************************************************************************
  184
  185   (load "~/transfer/fns/nagata.lisp")
  186
  187
```